**Exercise 3: Implementing the Builder Pattern**

* **Create a New Java Project:**
* You start by setting up a new Java project in your preferred Integrated Development Environment (IDE). This project will be named *BuilderPatternExample*.
* **Define a Product Class:**
* Create a class Computer with attributes like *CPU, RAM, Storage,* etc
* Each attribute represents a component of the computer and is declared as a private member variable within the *Computer* class.
* **Implement the Builder Class:**
  + A static nested class *Builder* is created inside the *Computer* class.
  + The *Builder* class contains the same attributes as the *Computer* class.
  + For each attribute in the *Computer* class, there is a corresponding setter method in the *Builder* class (e.g., *setCPU(String CPU)*).
  + The *Builder* class also includes a build() method that instantiates a *Computer* object using the *Builder's* attributes.
* **Implementing the Builder Pattern:**
* The *Computer* class has a private constructor that accepts a *Builder* object as a parameter.
* This constructor initializes the *Computer's* attributes using the values from the *Builder* object.
* The private constructor ensures that the only way to create a *Computer* instance is through the *Builder*
* **Testing the Builder Implementation:**
* A test class (typically within the *main* method) is created to demonstrate how to use the Builder pattern to create various *Computer* configurations.
* Multiple *Computer* objects are created with different attribute values using the Builder.
* The *toString()* method in the *Computer* class is overridden to provide a readable representation of the *Computer* objects, making it easier to verify the configurations.

In summary, the Builder Pattern is implemented by creating a nested *Builder* class within the *Computer* class, allowing for a flexible and readable way to construct *Computer* objects with optional parts. This approach ensures that the creation process is managed and attributes are set correctly, providing a clear and organized way to build complex objects.

Here is the code link – [link](https://github.com/Hyperstrom/Aniket-Pal_5017587/tree/main/WEEK-1/2.Design%20Patterns%20and%20Principles/Excercise-3)

Here is the output of the program --

